IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re application of:

TORMASOV *et al.*

Appl. No.: 09/918,032

Filed: July 30, 2001

For: **Distributed Network Data Storage System and Method**

Confirmation No.: 1026

Art Unit: 2151

Examiner: Patel, Dhairya A.

Atty. Docket: 2230.0390001/MBR/GSB

Declaration of Alexander Tormasov, Mikhail Khassine, Serguei Beloussov and Stanislav Protassov under 37 C.F.R. § 1.131

Commissioner for Patents
Washington, D.C. 20231

Sir:

The undersigned, Alexander Tormasov, Mikhail Khassine, Serguei Beloussov and Stanislav Protassov, declare and state that,

1. We are the inventors of the above-captioned application, U.S. Appl. No. 09/918,032, filed July 30, 2001.

2. We, the inventors, had completed our invention in a WTO country (specifically, at least one of the inventors, Serguei Beloussov, was, during the relevant time period, in the United States and Singapore), claimed in the subject application, prior to December 14, 2000, as evidenced by the following:

3. Exhibit A, entitled "ASPFS (project overview)", having an initial date prior to December 14, 2000, and a last edit date prior to December 14, 2000, which confirms the date of conception prior to the filing dates of both Boykin et al., U.S. Patent Publication No. 2002/0078461 and Lahr et al., U.S. Patent Publication No. 2002/0046405 (i.e., prior to December 14, 2000).

4. Exhibit B, a file entitled "config.h,v", and Exhibit C, a file entitled "fserv.cc,v," collectively represent a partial record of some of the work that was directed to actual reduction to practice of the invention. These two documents show a log of changes in a pilot implementation of a system embodying the ideas described in the instant patent application. The two exhibits confirm work on actual reduction to practice in Russia in the period from prior to December 14, 2000 through June 19, 2001 ("fserv.cc,v" file) and between February 15, 2001 through October 21, 2002 ("config.h,v" file).

5. The invention was also constructively reduced to practice on July 30, 2001, when the present non-provisional application was filed.

6. Thus, the invention was conceived prior to the filing dates of Boykin et al. and Lahr et al., and the inventors were working diligently on constructive and/or actual reduction to practice between the filing date of Boykin et al. and Lahr et al. and July 30, 2001, the filing date of this application.

7. As the persons signing below, we hereby declare that all statements made herein of our own knowledge are true and that all statements made on information and belief are believed to be true; and further that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under § 1001 of Title 18 of the United States Code, and that such willful false statements may jeopardize the validity of the application or any patent issue thereupon.

10.6.2005
Date

Alexander Tormasov

10.06.2005
Date

Mikhail Khassine

10/6/2005
Date

Serguei Beloussov

10.06.2005
Date

Stanislav Protassov

406122_1

Redundant Clustered Distributed Filesystem

RCDFS

Version 0.3

REDACTED

Revision History

Date	Version	Description	Author
[redacted]	0.0	Imported A.Tormasov document	Yuri Pudgorodsky
[redacted]	0.1	Design Draft	Yuri Pudgorodsky
[redacted]	0.2	Implementation Strategy Reworked	Yuri Pudgorodsky
[redacted]	0.3	Converted to HTML	Yuri Pudgorodsky

Table of Contents

---

1. Purpose

The purpose of this document is to provide short technical overview of RCDFS - Redundant Clustered Distributed Filesystem, base filesystem for ASP data Server of Project Wolf.

2. Overview

Data server should handle data. Data should be stored and delivered to place of request. Typically for such purposes modern OS use concept of file system (FS). Implementation of any FS be a very complex task, and, it usually hardly depend upon a underlying OS.

But typical set of requirements to local FS running on particular computer do not include features like high availability in cluster or effective support of distributed operations.

Proposed concept of Virtual Environments (VE) for each ASP end user requires that each instance of VE should have own set of files, including system ones.

Technically all operations inside VE could be done by local FS. For example, all write operations could be done locally.

Considering Linux as a platform for ASP we could mention, that all implementation of local FS do not have the following set of ASP platform requirements:

- **distributed access**
- **hierarchical uniform naming space**
- **high performance (fast access to any file in cluster)**
- **scalability support of 100s computers in network**
- **consistency journaling support and transactional features, including fast faults recovery**
- **fault tolerance - any computer could be switched off without loosing of access to any data in file system; any network connection could disappear**
- **self configurable - easy growth - any computer could be attached into network without user intervention into software (only some hardware connection should be done)**
- **security - ACL type access or (at least) UNIX style grouping security; optional encryption**

Some of these features have direct mapping in local FS abilities, some not. For example, VE from inside do not need distributed features, fault tolerance is not a question of functionality of software to be run inside VE, high performance - not a real question of typical ASP user. Mentioned problems are problems

of ASP - as service provider, quality of his service, maintainability, etc.

Otherwise, producing a new file system with all mentioned features could be very time- and resource- consuming task.

Reasonable compromise could be a development of storage server as an add-on to local file system.

Taking into account current features of Linux FS we could define the following set of data storage server requirements:

- **easy mapping to semantics of local FS on Linux - ASPdS should operate by the same concepts than local FS (at least, have superset of it implemented)**
- **easy integration of it with local FS - localFS have to obtain a new features by calling some ASPdS services, and have to do it effectively**
- **distributed access to files (any node in cluster could access it)**
- **scalability - close-to-linear extensibility in amount of nodes; addition of new node to cluster should not influence to overall performance**
- **controlled high availability of data - level of fault tolerance features should be controllable in range from localFS supported (usually nothing) up to transaction based (journalled) with all record operation immediately available**
- **maintainability - easiness of installation and handling (minimization of TCO)**
- **locking support for distributed access to file in global namespace**

Interesting, that we don't need a special locking feature for typical operation - each VE run under control of the single OS kernel on behalf of a given hardware unit, and support any locking from underlying OS. In case of distributed applications

they should utilize a couple of VEs on different nodes and use existing network-distributed SDK (MPI or PVM environment, for example).

In such a implementation we use a local FS as a file cache for global data storage.

Some applications (paraller database engines in cluster, for example) may want a high-availability distributed data storage. ASPdS with appropriate locking support and possible tuning of block allocation policy may provide such functionality.

What possibilities could be trade for these features? Probably we could use some CPU power, some additional disk space and some network performance. The actual penalty needs to be determined (measured) lately.

3. Design Principals

Here I describe main principles of organization of ASP cluster:

- **Equality of all nodes:**
  each node in cluster have the same control functions
- **Locality of all algorithms:**
  each node have information only about some adjacent neighbors it interested in and never have any global tables
- ***Reuse of existing file systems features:***
  each file stored on a local filesystem, application accesses data via a well balanced and robust implementations of native linux filesystems
- **Regulated redundancy of data:**

each file stored in form of pieces and amount of pieces could vary; each piece exist in the only exemplar in system (no traditional caching - we just cache additional amount of pieces) Pieces are stored on distributed nodes, providing high availability. Filesystems may be restored in case of absence of some pieces.

- **Versioned files:**
  each file stored in a set of transaction data (treat it as versions of file); and each version could not be changed both in size and data (instead we just generate a new transaction with new file version). There will be a user-controlled tools to access a previous version of a given file, list version information and restore a state of a file to some specified version.

- **Timestamped versions:**
  Transaction is marked with the current time and transaction ID - in order to provide and access to "snapshots" of filesystem for given time while maintaining unique id.

- **Uniformly distributed network loading:**
  we could move data in parallel manner from network to particular node (depending upon a topology) and thus utilize a network channel with minimal current network loading

- **Migration of information to the place of it's active utilization:**
  we could move pieces of file to nodes on which they are intensively used

- **Time should be synced on each nodes of cluster.**

4. Network Topology

Suggested physical network topology is a single segment within

a switched environment. For performance reasons node may want to know about its place in switched topology and the throughput of its network interface. However there are no direct restriction of a physical topology, it may be also a LAN with an arbitrary topology or even an global distributed environment. The only required ability is a presence of uniform broadcast or multicast group within a cluster.

Logically a node autoconfigure its place with the nearest neighbors to find active participants for redundant data storage. The filesystem daemons maintain permanently opened connection to node neighbors.

In case of a neighbor failure a parent node initiate a failover switch to another neighbor. After the restored availability of a previous neighbor it initiates a request to its parent node. The parent node may response with a switchover connection to a restored node or continue to work with the current neighbor, depending on the actual amount of its data available on one or another node.

During normal oparation nodes do not change its neighbors. The amount of neighbors used depends on required data redundancy (configuration parameter). The algorithm for automatic neighbors selection based on CPU/disk/network load needs been addressed lately.

5. Files representation

Each file consists of two parts:

- **locally stored data to provide fast access to the latest file version;**
- **remotely stored data to provide high availability and versioning.**

In case of unavailability of local representation (disk failure, node failure - VE restarted on different node, insufficient disk space, etc...) it will be retrieved from the remote representation.

Local representation is just a native file representation on a local filesystem. Only some parts of the file may been stored locally (for performance/bandwidth/disk space reasons). Local representation reuses the property of the local filesystem (i.e. journalling, very large file support, etc.). The proposed local filesystem for the ASP cluster should definitely include journalling support. Currently, there are four candidates on the role:

reiserfs	http://devlinux.com/projects/reiserfs/
ext	http://web.mit.edu/tytso/www/linux/
JFS	http://oss.software.ibm.com/developerworks/opensou
XFS	http://oss.sgi.com/projects/xfs/

Right now it is difficult to find out the best journalling FS for our needs, all the above filesystems are currently at the development and/or alpha testing stage. For today the reiserfs has been more thoroughly tested while IBM JVS has the most potentials due to it large history on AIX. Probably within the time constraints of the 2.4 linux kernel release (summer 2000) there will be more arguments for the choice.

Remote representation consists of set of extents: fixed size blocks. Block size could be variable. Files are assembled/reassembled in user space daemons. Not necessary to assemble whole file to provide access to them - it could be only small part of them.

Each block could have any number of versions. Versions are

numbered and sequenced by combination of timestamp and transaction ID number unique for each node (to avoid small resolution time and time sync problems). Main idea of such a numbering is a detection of last version of each file block to assemble.

Generally, remote file representation consists of set of blocks grouped in transactions.

6. Regulated redundancy and versioning

Each block stored in form: divided into N pieces, and any K pieces of them (K <= N) are enough to assemble initial block. Size of each block are minimized and equal to (size of block)/K with minimal overhead of some additional info in header. Appropriate mathematics are exists.

Typically file divided into at least N = (K+M) pieces, where M - amount of the nodes which could disappear simultaneously. Each block is stored on a separate remote node. There are K+M individual nodes with block pieces and the removal of M of them still gives an ability to restore block.

Maximal amount of possible pieces are restricted in the very initial stage of piece creation - and should be reasonable large (up to K*L where L - max nodes in cluster).

Each block piece exists in the only instance - we do not use any copies; instead, we provide an additional piece.

7. File access

File open:

- **Check for actuality of a local file copy. If the local copy is up-to-date, use it. Such check will use a network request only**

once-per-session of FS support daemons, so for typical system usage it will be a once per file from VE boot time.

- **If local copy not found:**
  - **translate logical name of file into physical ID of blocks (via directory data)**
  - **send a multicast request for blocks to network**
  - **after receiving number of replies that at least we could assemble one file extent release make a local copy of assembled data, and return open call with success.**
  - **send a block transfer packets (read-ahead)**
- **Think of:**
  - **Cache of a negative (file not found) replies.**

Read access (the same for one or more blocks not necessary to assemble all file locally!):

- **check for availability of current block to read in local filesystem**
- **if available read it and return**
- **if not available send search request, wait for response, and then send a pieces transfer request. Immediately after receiving of appropriate amount of pieces cancel all other transfers and assemble block in local cache; read it and return**

After receiving a search request each node have to return block with the latest (timestamp + transaction_counter).

Write requests depend upon a consistency requirements. Generally, we store file locally and queue disassemble request for remote storing. After the queuing transaction, we return success to a userspace from the write request.

As copy-on-write files, deleted files need to be marked in local VE cache too.

For files opened in synchronous write mode (O_SYNC) we could wait for remote storing procedure to complete physical write on remote neighbors. However this kind of redundancy seems a little paranoid for me so an intermediate solution is suggested: the write call will wait for physical operation on local file write request and queue transaction to be completed. In such way we ensure the locally store data has been gone to disk, and remote storing requests has been journalled locally to physical media.

In general, we can provide all levels of synchronous write support:

- **on synchronous operation;**
- **synchronous update of local FS transaction log (journalling local FS);**
- **synchronous update of local FS (UNIX O_SYNC);**
- **synchronous update of RCDFS remote store request;**
- **syncrounous RCDFS remote store (wait for data to be sent to neigbors);**
- **syncrounous RCDFS remote store, waiting for neigbors write to complete on physical media;**

The typical applications for each FS mode needs to be specified.

8. Garbage collection

Another significant question of such a versioning system is a garbage collection - removal of old block versions. Generally, we post all blocks to system only after commit operation for particular transaction. This means that if we want to remove any particular version of block we have to be sure that we could assemble next version of this block. The problem is that we have to be sure that ALL blocks participating in transaction available for assembling. Probably, purge request will be send by node posting transaction after successful placing of all pieces.

9. Pieces distribution

Pieces distribution algorithm have to place it on appropriate amount of nodes (in agreement with fault tolerance requirements), and provide some migration mechanism to minimize a network loading (just move each piece to node on which it will be used maximally intensively).

10. VE support

Each VE within a local node should receive its own separate filesystem namespace begin s from root. For the purpose of disk space optimization, we will provide a "copy-on-write" mechanism within a local node.

From global node namespace (not within VE) the given filesystem subtree is assigned for VE support. Within it, each VE receives its own subtree for the VE root namespace. Also, some another subtree has been assigned a status of "VE template".

For file read access, the VE personal subtree (/.ve/VE-ID/...) has been checked first. If file exists, it is accessed from here. If file here does not exists, VE template subtree has been checked. On success the special kind of link has been created within VE personal subtree. This link copies all file attributes while for the file content application is been redirected to VE template disk space.

For file read-write (modify) access, we extend this algorithm with copy on write policy: if application gets a template file and modifies its contents, the file content is actually copied into personal VE subtree replacing the special kind link.

File creation always occurred within personal VE subtree.

```
/--                                    (global namespace root)
|-- tmp
|-- usr
|       |-- bin
|       |-- lib
.......
|--.ve-template
             /--                       (template filesystem layo
             |-- bin

|--.ve                                 (mount point of VE suppor
       |-- 1
       |   /-- (root of VE with ID=1)
           |    |-- bin
           |
           |-- 21
           |    /--      (root of VEwith ID=21)
           |    |-- bin
```

11. Performance estimation

To be calculated.

12. Portability

In general, most VE implementations are kernel-dependent and could not be easily ported to another UNIX. But used open and highly modular approach could significantly simplify such a task.

The implementation of the cluster part of this project are highly portable and allow utilization not only different dialects of Unix on different hardware platforms, but also such platforms like Windows 2000.

13. Implementation Strategy

Like any network or distributed filesystems, RCDFS consists of a kernel-level FS modules and a set of user mode processes (daemons). The daemons run outside VE in the usermode

namespace of each node. On a later project stage, these daemons may been rewritten as a kernel-mode threads for performance reason.

Currently, the need for some daemons come mind:

- **ds-lookupd**
  - **node autoconfiguration**
  - **name lookup in the global cluster namespace**
- **ds-cored**
  - **handle RCDFS request for remote read and write requests**
  - **transaction id generation**
  - **handle read/write/delete block requests**
- **ds-supportd**
  - **collect usage statistic**
  - **garbage collection**

For broadcast traffic, IP multicasting may be used for easy access to an IP network with complex topology.

14. Things to be addressed

- **Name lookup algorithm: method for caching negative (file not found replies) locally.**
- **Disk / network bandwidth / cpu usage optimization criteria for choosing neighbors.**
- **Interaction^ with a linux buffer-cache: is it a task for our module or we just make a local file copy available and then redirect request to a local filesystem driver.**
- **Files stored locally only partial: method of redirecting local read request to remote transaction.**
- **Access to previous file versions: define semantics and user tools.**
- **Disk quotas.**
- **Implementation details of garbage collection.**

- **Global namespace support (cluster-visible).**
- **Distributed locking support.**

Document Owner: Yuri Pudgorodsky, ASPLinux yur@sw.mipt.ru
Base Document: Project Wolf, Techical Vision from Alex Tormasov, dated [redacted]

Redacted


```
head  1.13;
access;
symbols
      aspfs_0_9_1_dev:1.13
      aspfs_0_9_dev:1.12;
locks; strict;
comment     @ * @;


1.13
date  2002.10.21.20.07.14;    author dev; state Exp;
branches;
next  1.12;

1.12
date  2001.06.06.09.29.18;    author media;     state Exp;
branches;
next  1.11;

1.11
date  2001.05.14.13.48.45;    author media;     state Exp;
branches;
next  1.10;

1.10
date  2001.05.02.12.12.18;    author dev; state Exp;
branches;
next  1.9;

1.9
date  2001.04.13.09.23.57;    author media;     state Exp;
branches;
next  1.8;

1.8
date  2001.04.12.09.51.01;    author media;     state Exp;
branches;
next  1.7;

1.7
date  2001.04.10.14.55.41;    author media;     state Exp;
branches;
next  1.6;

1.6
date  2001.03.31.14.24.19;    author media;     state Exp;
branches;
next  1.5;

1.5
date  2001.03.30.10.31.07;    author media;     state Exp;
branches;
next  1.4;

1.4
date  2001.03.22.09.14.13;    author media;     state Exp;
branches;
next  1.3;

1.3
date  2001.03.12.11.18.26;    author media;     state Exp;
branches;
next  1.2;
```

```
1.2
date  2001.02.26.12.15.11;    author media;     state Exp;
branches;
next  1.1;

1.1
date  2001.02.15.14.04.50;    author media;     state Exp;
branches;
next  ;


desc
@@


1.13
log
@Added WorkDir
@
text
@/*
 * Copyright (C) SWsoft 2000-2001
 *
 * Author:  Ruslan Iljin <media@@www.rt.mipt.ru>
 *
 * $Id: config.h,v 1.12 2001/06/06 09:29:18 media Exp $
 *
 * Description: headers for config file processing
 *
 */

#ifndef __TOPD_CONFIG_H__
#define __TOPD_CONFIG_H__

#ifdef __cplusplus
extern "C" {
#endif

#ifdef DEFINE_CONFIG_VARIABLES
  #define DECLARE
  #define DEFAULT(x) =x;
#else
  #define DEFAULT(x)
  #ifdef __cplusplus
    #define DECLARE extern "C"
  #else
    #define DECLARE extern
  #endif
#endif

int parse_config(const char* conf_name, int looketc);

DECLARE void add_neighbour_from_config(char *neigh);

DECLARE char* BindToAddress DEFAULT("0.0.0.0");
DECLARE char* WorkDir DEFAULT("~/storage");

DECLARE char* CoordAddress DEFAULT(NULL);

DECLARE int   Daemon                DEFAULT(1);

/* timeouts */
DECLARE int   TOPD_TIMEOUT_RECONNECT     DEFAULT(30);
```

```
DECLARE int   TOPD_TIMEOUT_STAGE_0       DEFAULT(5);

DECLARE int   TOPD_TIMEOUT_WRITE_STAGE_1 DEFAULT(20);
DECLARE int   TOPD_TIMEOUT_WRITE_STAGE_2 DEFAULT(20);

DECLARE int   TOPD_TIMEOUT_READ_STAGE_1  DEFAULT(60);
DECLARE int   TOPD_TIMEOUT_READ_STAGE_2  DEFAULT(60);

DECLARE int   TOPD_TIMEOUT_SNAP_STAGE_1  DEFAULT(15);
DECLARE int   TOPD_TIMEOUT_SNAP_STAGE_2  DEFAULT(20);

DECLARE int   TOPD_TIMEOUT_DIRREC_STAGE_1 DEFAULT(15);

DECLARE int   TOPD_TIMEOUT_BLOCK0_STAGE_1 DEFAULT(15);

DECLARE int   TOPD_TIMEOUT_PROXY         DEFAULT(10);

#ifdef __cplusplus
}            // extern "C"
#endif

#undef _TOPD_READ_ALL_SLICES_     /* read all available slices? if undef read
only required number of slices */
#undef _TOPD_REMOVE_AFTER_READ_   /* remove snap entry after reading? */
#undef _TOPD_LOOP_PING_           /* continue pinging neighbours for dynamic
table? */

#endif
@


1.12
log
@*** empty log message ***
@
text
@d6 1
a6 1
 * $Id: config.h,v 1.11 2001/05/14 13:48:45 media Exp $
d36 1
@


1.11
log
@fixed many bugs in lists
@
text
@d6 1
a6 1
 * $Id: config.h,v 1.10 2001/05/02 12:12:18 dev Exp $
d56 2
@


1.10
log
@Global code review.
TFS_assert -> ASSERT
removed all tfsLog functions. debug() is used instead
SWSoft -> SWsoft :)
@
text
@d6 1
```

```
a6 1
 * $Id: config.h,v 1.9 2001/04/13 09:23:57 media Exp $
d33 2
d37 1
a37 1
DECLARE char* CoordAddress DEFAULT("0.0.0.0");
@


1.9
log
@remove trailing spaces
@
text
@d2 1
a2 1
 * Copyright (C) SWSoft 2000-2001
d4 3
a6 1
 * $Id: config.h,v 1.8 2001/04/12 09:51:01 media Exp $
a8 2
 *
 * Author:  Ruslan Iljin <media@@www.rt.mipt.ru>
@


1.8
log
@header files description
@
text
@d4 1
a4 1
 * $Id: config.h,v 1.41 2001/04/12 08:26:34 media Exp $
d11 1
a11 1

@


1.7
log
@code review
@
text
@d1 11
d39 1
a39 1
/* timeouts */
d44 2
a45 2
DECLARE int   TOPD_TIMEOUT_WRITE_STAGE_1 DEFAULT(20);
DECLARE int   TOPD_TIMEOUT_WRITE_STAGE_2 DEFAULT(20);
d47 2
a48 2
DECLARE int   TOPD_TIMEOUT_READ_STAGE_1  DEFAULT(60);
DECLARE int   TOPD_TIMEOUT_READ_STAGE_2  DEFAULT(60);
d50 2
a51 2
DECLARE int   TOPD_TIMEOUT_SNAP_STAGE_1  DEFAULT(15);
DECLARE int   TOPD_TIMEOUT_SNAP_STAGE_2  DEFAULT(20);
d53 1
a53 1
DECLARE int   TOPD_TIMEOUT_DIRREC_STAGE_1 DEFAULT(15);
```

```
@


1.6
log
@changes in dyn tables
@
text
@d28 17
a44 1
DECLARE int   TOPD_TIMEOUT_RECONNECT DEFAULT(30);
@


1.5
log
@changes for new analize functions
@
text
@d34 3
a36 3
#define _TOPD_READ_ALL_SLICES_     0    /* read all available slices? if 0 read
only required number of slices */
#define _TOPD_REMOVE_AFTER_READ_   0    /* remove snap entry after reading? */
#define _TOPD_LOOP_PING_           0    /* continue pinging neighbours for
dynamic table? */
@


1.4
log
@*** empty log message ***
@
text
@d1 2
a2 2
#ifndef __CONFIG_H__
#define __CONFIG_H__
d33 4
@


1.3
log
@*** empty log message ***
@
text
@d28 2
@


1.2
log
@*** empty log message ***
@
text
@d24 2
@


1.1
log
@*** empty log message ***
@
```

```
text
@a22 12
DECLARE char* TopdAddress   DEFAULT("127.0.0.1");

DECLARE char* WorkDir DEFAULT("~/clroot/");
DECLARE char* TmpDir DEFAULT("/tmp/");

DECLARE int   PutFileMaxBloxPerStep DEFAULT(100);
DECLARE int   GetFileMaxBloxPerStep DEFAULT(100);

DECLARE int   TimeOutPut0Block      DEFAULT(5);
DECLARE int   TimeOutGetSnap        DEFAULT(5);

DECLARE int   TimeOutTopdReconnect  DEFAULT(5);
@
```


```
head  1.34;
access;
symbols
      aspfs_0_9_1_dev:1.34
      aspfs_0_9_dev:1.34;
locks; strict;
comment     @// @;


1.34
date  2001.06.19.13.15.53;    author media;     state Exp;
branches;
next  1.33;

1.33
date  2001.06.06.09.29.18;    author media;     state Exp;
branches;
next  1.32;

1.32
date  2001.06.01.12.22.23;    author media;     state Exp;
branches;
next  1.31;

1.31
date  2001.06.01.10.22.34;    author media;     state Exp;
branches;
next  1.30;

1.30
date  2001.05.02.12.12.18;    author dev; state Exp;
branches;
next  1.29;

1.29
date  2001.04.25.16.18.56;    author media;     state Exp;
branches;
next  1.28;

1.28
date  2001.04.20.10.46.27;    author media;     state Exp;
branches;
next  1.27;

1.27
date  2001.04.20.10.26.04;    author media;     state Exp;
branches;
next  1.26;

1.26
date  2001.04.20.09.23.00;    author media;     state Exp;
branches;
next  1.25;

1.25
date  2001.04.19.11.10.46;    author media;     state Exp;
branches;
next  1.24;

1.24
date  2001.04.19.08.22.10;    author media;     state Exp;
branches;
next  1.23;
```

```
1.23
date  2001.04.13.09.23.57;    author media;     state Exp;
branches;
next  1.22;

1.22
date  2001.04.12.11.37.14;    author media;     state Exp;
branches;
next  1.21;

1.21
date  2001.04.12.11.35.27;    author media;     state Exp;
branches;
next  1.20;

1.20
date  2001.04.12.11.33.26;    author media;     state Exp;
branches;
next  1.19;

1.19
date  2001.04.12.10.42.10;    author media;     state Exp;
branches;
next  1.18;

1.18
date  2001.04.12.10.34.20;    author media;     state Exp;
branches;
next  1.17;

1.17
date  2001.04.12.10.26.35;    author media;     state Exp;
branches;
next  1.16;

1.16
date  2001.04.04.10.33.33;    author media;     state Exp;
branches;
next  1.15;

1.15
date  2001.03.30.12.43.36;    author media;     state Exp;
branches;
next  1.14;

1.14
date  2001.03.23.15.29.23;    author sinvv;     state Exp;
branches;
next  1.13;

1.13
date  2001.02.23.11.15.18;    author media;     state Exp;
branches;
next  1.12;

1.12
date  2001.02.22.14.52.34;    author media;     state Exp;
branches;
next  1.11;

1.11
date  2001.01.30.17.48.43;    author media;     state Exp;
branches;
next  1.10;
```

```
1.10
date  2001.01.30.17.29.44;    author media;     state Exp;
branches;
next  1.9;

1.9
date  [redacted].11.43.19;    author media;     state Exp;
branches;
next  1.8;

1.8
date  [redacted].09.38.18;    author media;     state Exp;
branches;
next  1.7;

1.7
date  [redacted].14.40.12;    author media;     state Exp;
branches;
next  1.6;

1.6
date  [redacted].15.11.41;    author media;     state Exp;
branches;
next  1.5;

1.5
date  [redacted].14.20.48;    author media;     state Exp;
branches;
next  1.4;

1.4
date  [redacted].13.01.11;    author media;     state Exp;
branches;
next  1.3;

1.3
date  [redacted].13.18.39;    author media;     state Exp;
branches;
next  1.2;

1.2
date  [redacted].08.57.38;    author media;     state Exp;
branches;
next  1.1;

1.1
date  [redacted].13.21.27;    author media;     state Exp;
branches;
next  ;


desc
@@


1.34
log
@*** empty log message ***
@
text
@/*
 * Copyright (C) SWsoft 2000-2001
 *
```

REDACTED

```
 * Author:  Ruslan Iljin <media@@www.rt.mipt.ru>
 *
 * $Id: fserv.cc,v 1.33 2001/06/06 09:29:18 media Exp $
 *
 * Description: this file contains interface to file server
 *
 */

#include "topd.h"

/*
 * function      :   tfsTopd::fserv_send
 * prototype in  :   topd.h
 * description   :   send packet to client channel
 * parameters    :   p  - packet to send
 * return        :   int - send count
 * errors        :   TFS_E_TOPD_FSERV_WRONG
 */
int tfsTopd::fserv_send(tfsPKT *p)
{
  int count=0;

  INFUNC(tfsTopd::fserv_send);
  if (!C_fserv) {
    PRINTD2("fileserver_send: there is no fserv");
    OUTFUNCINT(TFS_E_TOPD_FSERV_WRONG);
  }

  if  ((C_fserv->state == tfsChannel::CONNECTED)||(C_fserv->state ==
tfsChannel::OPENED))
  {
    PRINTD2("Packet sent to fserv");
    C_fserv->PutQ(p);
    count++;
  }
  OUTFUNCINT(count);
}

/*
 * function      :   fserv_common_request
 * prototype in  :   topd.h
 * description   :   send snap, read, dirrec, write request to fserv
 * parameters    :   reqid - RequestID of snap request
 *                   buff  - address of data buffer
 *                   length   - length of data buffer
 *                   req_type - request type
 * return        :   int - result of fserv_send
 * errors        :   TFS_E_TOPD_NO_MEMORY
 */
int tfsTopd::fserv_common_request(tfsRequestID reqid, char* buff, int length,
int req_type)
{
  int rc;

  tfsPktType p_type[7]={
  TFS_MSG_T2F_SNAP_REQ,
  TFS_MSG_T2F_SLICE_READ_REQ,
  TFS_MSG_T2F_DIRRECORD_LIST_REQ,
  TFS_MSG_T2F_BLOCK0_LIST_REQ,
  TFS_MSG_T2F_SLICE_WRITE_REQ,
  TFS_MSG_T2F_ZEROBLOCK_WRITE_REQ,
  TFS_MSG_T2F_DIRRECORD_WRITE_REQ};

  INFUNC(tfsTopd::fserv_common_request);
```

```
  if (req_type==TOPD_SNAP_REQUEST) PRINTD2("send snap request to fserv");
  if (req_type==TOPD_READ_REQUEST) PRINTD2("send read request to fserv");
  if (req_type==TOPD_DIRREC_REQUEST) PRINTD2("send directory record list request
to fserv");
  if (req_type==TOPD_BLOCK0_REQUEST) PRINTD2("send block0 list request to
fserv");
  if (req_type==TOPD_SLICE_WRITE) PRINTD2("send slice write request to fserv");
  if (req_type==TOPD_ZEROBLOCK_WRITE) PRINTD2("send zeroblock write request to
fserv");
  if (req_type==TOPD_DIRREC_WRITE) PRINTD2("send directory record write request
to fserv");

  tfsPKT *p = new tfsPKT(p_type[req_type],length,buff,reqid);
  if (!p) {
    PRINTD("ERROR: not enough memory");
    OUTFUNCINT(TFS_E_TOPD_NO_MEMORY);
  }
  rc=fserv_send(p);

  if (TFS_E_CHECK(rc)) rc=0;

  OUTFUNCINT(rc);
}

/*
 * function      :   fserv_search_reply
 * prototype in  :   topd.h
 * description   :   processing snap reply from fserv
 * parameters    :   p - packet with reply
 * return        :   int - result of initiate function
 * errors        :   apropriate to initiate function
 */
int tfsTopd::fserv_search_reply(tfsPKT *p)
{
  int rc;

  INFUNC(tfsTopd::fserv_search_reply);
  PRINTD2("got slice search reply from fserv");
  if (header_get_hops_number(p->data, p->length)) rc=initiate_search_reply(p,
TOPD_SLICE_SEARCH_REPLY);
  else rc=search_reply(p, get_neighbour_number(myself),
TOPD_SLICE_SEARCH_REPLY);
  OUTFUNCINT(rc);
}

/*
 * function      :   fserv_read_reply
 * prototype in  :   topd.h
 * description   :   processing read reply from fserv
 * parameters    :   p - packet with reply
 * return        :   int - result of initiate function
 * errors        :   apropriate to initiate function
 */
int tfsTopd::fserv_read_reply(tfsPKT *p)
{
  int rc;

  INFUNC(tfsTopd::fserv_read_reply);
  PRINTD2("got slice read reply from fserv");
  if (header_get_hops_number(p->data, p->length)) rc=initiate_search_reply(p,
TOPD_SLICE_READ_REPLY);
  else rc=search_reply(p, get_neighbour_number(myself), TOPD_SLICE_READ_REPLY);
  OUTFUNCINT(rc);
```

```
}

/*
 * function      :   fserv_dirrecord_reply
 * prototype in :   topd.h
 * description   :   processing dirrecord reply from fserv
 * parameters    :   p - packet with reply
 * return        :   int - result of initiate function
 * errors        :   apropriate to initiate function
 */
int tfsTopd::fserv_dirrecord_reply(tfsPKT *p)
{
  int rc;

  INFUNC(tfsTopd::fserv_dirrecord_reply);
  PRINTD2("got dirrec reply from fserv");
  if (header_get_hops_number(p->data, p->length)) rc=initiate_search_reply(p,
TOPD_DIRREC_SEARCH_REPLY);
  else rc=search_reply(p, get_neighbour_number(myself),
TOPD_DIRREC_SEARCH_REPLY);
  OUTFUNCINT(rc);
}

/*
 * function      :   fserv_block0_reply
 * prototype in :   topd.h
 * description   :   processing block0 reply from fserv
 * parameters    :   p - packet with reply
 * return        :   int - result of initiate function
 * errors        :   apropriate to initiate function
 */
int tfsTopd::fserv_block0_reply(tfsPKT *p)
{
  int rc;

  INFUNC(tfsTopd::fserv_block0_reply);
  PRINTD2("got block0 reply from fserv");
  if (header_get_hops_number(p->data, p->length)) rc=initiate_search_reply(p,
TOPD_BLOCK0_SEARCH_REPLY);
  else rc=search_reply(p, get_neighbour_number(myself),
TOPD_BLOCK0_SEARCH_REPLY);
  OUTFUNCINT(rc);
}
@


1.33
log
@*** empty log message ***
@
text
@d6 1
a6 1
 * $Id: fserv.cc,v 1.32 2001/06/01 12:22:23 media Exp $
a161 1

@


1.32
log
@*** empty log message ***
@
text
```

```
@d6 1
a6 1
 * $Id: fserv.cc,v 1.31 2001/06/01 10:22:34 media Exp $
d56 1
a56 1
  tfsPktType p_type[6]={
d60 1
d70 1
d91 1
a91 3
 * parameters    :    reqid - RequestID of snap request
 *                    buff  - address of data buffer
 *                    length   - length of data buffer
d110 1
a110 3
 * parameters    :    reqid - RequestID of snap request
 *                    buff  - address of data buffer
 *                    length   - length of data buffer
d129 1
a129 3
 * parameters    :    reqid - RequestID of snap request
 *                    buff  - address of data buffer
 *                    length   - length of data buffer
d141 19
@


1.31
log
@*** empty log message ***
@
text
@d6 1
a6 1
 * $Id: fserv.cc,v 1.30 2001/05/02 12:12:18 dev Exp $
d95 1
a95 1
int tfsTopd::fserv_search_reply(tfsRequestID reqid, char* buff, int length)
d101 2
a102 2
  if (header_get_hops_number(buff, length)) rc=initiate_search_reply(reqid,
buff, length, TOPD_SLICE_SEARCH_REPLY);
  else rc=search_reply(reqid, get_neighbour_number(myself), buff, length,
TOPD_SLICE_SEARCH_REPLY);
d116 1
a116 1
int tfsTopd::fserv_read_reply(tfsRequestID reqid, char* buff, int length)
d122 2
a123 2
  if (header_get_hops_number(buff, length)) rc=initiate_search_reply(reqid,
buff, length, TOPD_SLICE_READ_REPLY);
  else rc=search_reply(reqid, get_neighbour_number(myself), buff, length,
TOPD_SLICE_READ_REPLY);
d137 1
a137 1
int tfsTopd::fserv_dirrecord_reply(tfsRequestID reqid, char* buff, int length)
d143 2
a144 2
  if (header_get_hops_number(buff, length)) rc=initiate_search_reply(reqid,
buff, length, TOPD_DIRREC_SEARCH_REPLY);
  else rc=search_reply(reqid, get_neighbour_number(myself), buff, length,
TOPD_DIRREC_SEARCH_REPLY);
@
```

```
1.30
log
@Global code review.
TFS_assert -> ASSERT
removed all tfsLog functions. debug() is used instead
SWSoft -> SWsoft :)
@
text
@d6 1
a6 1
 * $Id: fserv.cc,v 1.29 2001/04/25 16:18:56 media Exp $
d79 2
@


1.29
log
@*** empty log message ***
@
text
@d2 1
a2 1
 * Copyright (C) SWSoft 2000-2001
d4 3
a6 1
 * $Id: fserv.cc,v 1.28 2001/04/20 10:46:27 media Exp $
a8 2
 *
 * Author:  Ruslan Iljin <media@@www.rt.mipt.ru>
@


1.28
log
@join read request
@
text
@d4 1
a4 1
 * $Id: fserv.cc,v 1.27 2001/04/20 10:26:04 media Exp $
d100 1
a100 1
  else rc=search_reply(reqid, get_neighbour_number(myself), buff, length,
TOPD_DIRREC_SEARCH_REPLY);
@


1.27
log
@final search request imple,entation
@
text
@d4 1
a4 1
 * $Id: fserv.cc,v 1.26 2001/04/20 09:23:00 media Exp $
d120 2
a121 2
  if (header_get_hops_number(buff, length))
rc=initiate_slice_transfer_reply(reqid, buff, length);
  else rc=slice_transfer_reply(reqid, get_neighbour_number(myself), buff,
length);
@
```

```
1.26
log
@join search proc
@
text
@d4 1
a4 1
 * $Id: fserv.cc,v 1.25 2001/04/19 11:10:46 media Exp $
d100 1
a100 1
  else rc=slice_search_reply(reqid, get_neighbour_number(myself), buff, length);
d142 1
a142 1
  else rc=dirrecord_reply(reqid, get_neighbour_number(myself), buff, length);
@


1.25
log
@add TOPD prefix to const
@
text
@d4 1
a4 1
 * $Id: fserv.cc,v 1.24 2001/04/19 08:22:10 media Exp $
d99 1
a99 1
  if (header_get_hops_number(buff, length))
rc=initiate_slice_search_reply(reqid, buff, length);
d141 1
a141 1
  if (header_get_hops_number(buff, length)) rc=initiate_dirrecord_reply(reqid,
buff, length);
@


1.24
log
@remove neighbour address attachment
@
text
@d4 1
a4 1
 * $Id: fserv.cc,v 1.23 2001/04/13 09:23:57 media Exp $
d66 6
a71 6
  if (req_type==SNAP_REQUEST) PRINTD2("send snap request to fserv");
  if (req_type==READ_REQUEST) PRINTD2("send read request to fserv");
  if (req_type==DIRREC_REQUEST) PRINTD2("send directory record list request to
fserv");
  if (req_type==SLICE_WRITE) PRINTD2("send slice write request to fserv");
  if (req_type==ZEROBLOCK_WRITE) PRINTD2("send zeroblock write request to
fserv");
  if (req_type==DIRREC_WRITE) PRINTD2("send directory record write request to
fserv");
@


1.23
log
@remove trailing spaces
@
text
```

```
@d4 1
a4 1
 * $Id: fserv.cc,v 1.22 2001/04/12 11:37:14 media Exp $
d100 1
a100 1
  else rc=slice_search_reply(reqid, myself->Address, buff, length);
d121 1
a121 1
  else rc=slice_transfer_reply(reqid, myself->Address, buff, length);
d142 1
a142 1
  else rc=dirrecord_reply(reqid, myself->Address, buff, length);
@


1.22
log
@change no memory debug level
@
text
@d4 1
a4 1
 * $Id: fserv.cc,v 1.21 2001/04/12 11:35:27 media Exp $
d77 1
a77 1
  }
@


1.21
log
@no memory check
@
text
@d4 1
a4 1
 * $Id: fserv.cc,v 1.20 2001/04/12 11:33:26 media Exp $
d75 1
a75 1
    PRINTD1("ERROR: not enough memory");
@


1.20
log
@*** empty log message ***
@
text
@d4 1
a4 1
 * $Id: fserv.cc,v 1.19 2001/04/12 10:42:10 media Exp $
d74 4
@


1.19
log
@fserv desription
@
text
@d4 1
a4 1
 * $Id: fserv.cc,v 1.18 2001/04/12 10:34:20 media Exp $
d64 1
```

```
a64 1
  INFUNC(tfsTopd::fserv_write_request);
@


1.18
log
@*** empty log message ***
@
text
@d4 1
a4 1
 * $Id: fserv.cc,v 1.17 2001/04/12 10:26:35 media Exp $
d41 11
d55 1
d65 1
d72 1
d75 1
d79 10
d100 31
a141 10
int tfsTopd::fserv_read_reply(tfsRequestID reqid, char* buff, int length)
{
  int rc;

  INFUNC(tfsTopd::fserv_read_reply);
  PRINTD2("got slice read reply from fserv");
  if (header_get_hops_number(buff, length))
rc=initiate_slice_transfer_reply(reqid, buff, length);
  else rc=slice_transfer_reply(reqid, myself->Address, buff, length);
  OUTFUNCINT(rc);
}
@


1.17
log
@header files description
@
text
@d4 1
a4 1
 * $Id: fserv.cc,v 1.16 2001/04/04 10:33:33 media Exp $
d55 1
a55 1
  if (req_type==DIRREC_REQUST) PRINTD2("send directory record list request to
fserv");
@


1.16
log
@code review
@
text
@d4 1
a4 1
 * Author:    Ruslan Iljin <media@@www.rt.mipt.ru>
d6 1
a6 1
 * $Id: fserv.cc,v 1.15 2001/03/30 12:43:36 media Exp $
d8 1
a8 6
 */
```

```
/*
 * fserv.cc
 *
 * This file contains interface to file server
d14 8
d28 1
a28 1
    PRINTD2("fileserver_send: there is no fserv\n");
d34 1
a34 1
    PRINTD2("Packet sent to fserv\n");
d41 1
a41 1
int tfsTopd::fserv_write_request(tfsRequestID reqid, char* buff, int length, int
req_type)
d44 7
a50 1
  tfsPktType p_type[3]={TFS_MSG_T2F_ZEROBLOCK_WRITE_REQ,
TFS_MSG_T2F_SLICE_WRITE_REQ, TFS_MSG_T2F_DIRRECORD_WRITE_REQ};
d53 6
a58 3
  if (req_type==SLICE_WRITE) PRINTD2("send slice write request to fserv\n");
  if (req_type==ZEROBLOCK_WRITE) PRINTD2("send zeroblock write request to
fserv\n");
  if (req_type==DIRREC_WRITE) PRINTD2("send directory record write request to
fserv\n");
a63 22
int tfsTopd::fserv_search_request(tfsRequestID reqid, char* buff, int length)
{
  int rc;

  INFUNC(tfsTopd::fserv_search_request);
  PRINTD2("send slice search request to fserv\n");
  tfsPKT *p = new tfsPKT(TFS_MSG_T2F_SNAP_REQ,length,buff,reqid);
  rc=fserv_send(p);
  OUTFUNCINT(rc);
}

int tfsTopd::fserv_dirrecord_request(tfsRequestID reqid, char* buff, int length)
{
  int rc;

  INFUNC(tfsTopd::fserv_dirrecord_request);
  PRINTD2("send dirrec request to fserv\n");
  tfsPKT *p = new tfsPKT(TFS_MSG_T2F_DIRRECORD_LIST_REQ,length,buff,reqid);
  rc=fserv_send(p);
  OUTFUNCINT(rc);
}

d69 1
a69 1
  PRINTD2("got slice search reply from fserv\n");
d80 1
a80 1
  PRINTD2("got dirrec reply from fserv\n");
a85 11
int tfsTopd::fserv_read_request(tfsRequestID reqid, char* buff, int length)
{
  int rc;

  INFUNC(tfsTopd::fserv_read_request);
  PRINTD2("send slice read request to fserv\n");
  tfsPKT *p = new tfsPKT(TFS_MSG_T2F_SLICE_READ_REQ,length,buff,reqid);
```

```
  rc=fserv_send(p);
  OUTFUNCINT(rc);
}

d91 1
a91 1
  PRINTD2("got slice read reply from fserv\n");
@


1.15
log
@removed tabs spaces
@
text
@d4 1
a4 1
 * Author:  Ruslan Iljin <media@@www.rt.mipt.ru>
d6 1
a6 1
 * $Id: fserv.cc,v 1.14 2001/03/23 15:29:23 sinvv Exp $
d10 2
a11 2
/*
 * fserv.cc
d23 1
d26 1
a26 1
    return -1;
d29 2
a30 2
  if  ((C_fserv->state == tfsChannel::CONNECTED)||(C_fserv->state ==
tfsChannel::OPENED))
  {
d34 2
a35 2
  }
  return count;
d40 1
d42 2
d47 3
a49 2
  tfsPKT *p = new tfsPKT(p_type[req_type],length,buff,reqid);
  fserv_send(p);
d51 1
a51 1

d54 7
a60 3
  PRINTD2("send slice search request to fserv\n");
  tfsPKT *p = new tfsPKT(TFS_MSG_T2F_SNAP_REQ,length,buff,reqid);
  fserv_send(p);
d65 7
a71 3
  PRINTD2("send dirrec request to fserv\n");
  tfsPKT *p = new tfsPKT(TFS_MSG_T2F_DIRRECORD_LIST_REQ,length,buff,reqid);
  fserv_send(p);
d76 7
a82 3
  PRINTD2("got slice search reply from fserv\n");
  if (header_get_hops_number(buff, length)) initiate_slice_search_reply(reqid,
buff, length);
  else slice_search_reply(reqid, myself->Address, buff, length);
d87 7
```

```
a93 3
  PRINTD2("got dirrec reply from fserv\n");
  if (header_get_hops_number(buff, length)) initiate_dirrecord_reply(reqid,
buff, length);
  else dirrecord_reply(reqid, myself->Address, buff, length);
d98 7
a104 3
  PRINTD2("send slice read request to fserv\n");
  tfsPKT *p = new tfsPKT(TFS_MSG_T2F_SLICE_READ_REQ,length,buff,reqid);
  fserv_send(p);
d109 7
a115 3
  PRINTD2("got slice read reply from fserv\n");
  if (header_get_hops_number(buff, length)) initiate_slice_transfer_reply(reqid,
buff, length);
  else slice_transfer_reply(reqid, myself->Address, buff, length);
@


1.14
log
@Change copyright sinvv@@
@
text
@d6 1
a6 1
 * $Id: /home/cvs/aspfs/topd/fserv.cc,v 1.13 2001/02/23 11:15:18 media Exp $
d24 2
a25 2
      PRINTD2("fileserver_send: there is no fserv\n");
      return -1;
d29 5
a33 5
        {
            PRINTD2("Packet sent to fserv\n");
            C_fserv->PutQ(p);
            count++;
        }
@


1.13
log
@*** empty log message ***
@
text
@d2 1
d4 1
a4 1
 * Copyright (C) SWSoft 1999-2000
d6 1
a6 6
 * Author:              Ruslan Iljin
 * E-mail:              media@@www.rt.mipt.ru
 *
 * $Header: /home/cvs/aspfs/topd/fserv.cc,v 1.12 2001/02/22 14:52:34 media Exp $
 *
 * Last correct: [redacted]
@

REDACTED

1.12
log
@*** empty log message ***
```

```
@
text
@d8 1
a8 1
 * $Header: /home/cvs/aspfs/topd/fserv.cc,v 1.11 2001/01/30 17:48:43 media Exp $
d58 1
a58 1
int tfsTopd::fserv_params_request(tfsRequestID reqid, char* buff, int length)
d60 2
a61 2
  PRINTD2("send params request to fserv\n");
  tfsPKT *p = new tfsPKT(TFS_MSG_T2F_PARAMS_REQ,length,buff,reqid);
d72 1
a72 1
int tfsTopd::fserv_params_reply(tfsRequestID reqid, char* buff, int length)
d74 3
a76 3
  PRINTD2("got params reply from fserv\n");
  if (header_get_hops_number(buff, length)) initiate_params_reply(reqid, buff,
length);
  else params_reply(reqid, myself->Address, buff, length);
@


1.11
log
@*** empty log message ***
@
text
@d8 1
a8 1
 * $Header: /home/cvs/aspfs/topd/fserv.cc,v 1.10 2001/01/30 17:29:44 media Exp $
d43 1
a43 1
  tfsPktType p_type[2]={TFS_MSG_T2F_ZEROBLOCK_WRITE_REQ,
TFS_MSG_T2F_SLICE_WRITE_REQ};
d45 2
a46 1
   else PRINTD2("send zeroblock write request to fserv\n");
@


1.10
log
@*** empty log message ***
@
text
@d8 1
a8 1
 * $Header: /home/cvs/aspfs/topd/fserv.cc,v 1.9 2000/12/05 11:43:19 media Exp $
d54 7
@


1.9
log
@*** empty log message ***
@
text
@d8 1
a8 1
 * $Header: /home/cvs/aspfs/topd/fserv.cc,v 1.8 [redacted] 09:38:18 media Exp $
d62 7
@
```

REDACTED

```
1.8
log
@*** empty log message ***
@
text
@d8 1
a8 1
 * $Header: /usr/virtual/cvsroot/torfs/topd/fserv.cc,v 1.3 2000/11/29 12:21:34
media Exp $
d35 1
a35 1
            C_fserv->PutQ(p->copy());
@


1.7
log
@*** empty log message ***
@
text
@@


1.6
log
@*** empty log message ***
@
text
@@


1.5
log
@*** empty log message ***
@
text
@@


1.4
log
@*** empty log message ***
@
text
@@


1.3
log
@*** empty log message ***
@
text
@@


1.2
log
@*** empty log message ***
@
text
@d8 1
a8 1
```

```
 * $Header: /usr/virtual/cvsroot/torfs/topd/fserv.cc,v 1.2 [redacted] 10:33:39
media Exp $
d44 2
a45 2
  PRINTD2("send slice write request to fserv\n");
//  if (type==TFS_MSG_C2T_BLOCK_WRITE_REQ) type=TFS_MSG_T2F_BLOCK_WRITE_REQ;
a61 7
}

int tfsTopd::fserv_write_reply(tfsRequestID reqid, char* buff, int length)
{
  PRINTD2("got write reply from fserv\n");
  if (header_get_hops_number(buff, length)) initiate_write_reply(reqid, buff,
length);
  else write_reply(reqid, myself->Address, buff, length);
@


1.1
log
@*** empty log message ***
@
text
@@
```

REDACTED